Towards Super-Scalable Multicast *

Jörg Liebeherr† *Bhupinder S. Sethi*‡

† Polytechnic University
Department of Electrical Engineering
6 MetroTech Center
Brooklyn, NY 11201
Phone: (718) 260-3493, Fax: (718) 260-3074
email: `jorg@catt.poly.edu`

‡ Department of Computer Science
University of Virginia
Charlottesville, VA 22903
email: `bss4k@cs.virginia.edu`

Polytechnic University, CATT, Technical Report 98-121

Abstract

Large-Scale multicast applications for the Internet require the availability of multicast protocols that can support multicast groups with many thousand simultaneous members. For this it is crucial that there exist mechanisms to efficiently exchange control information between the members of a group. In this paper, we present a new approach for distributing control information within a multicast group which increases the scalability of multicast applications. Multicast group members are organized as a logical *n-dimensional hypercube*, and all control information is transmitted along the edges of the hypercube. We analyze the scalability of the hypercube control topology and compare it with tree-based approaches. We show that the hypercube balances the load per member for processing control information better than existing topologies. We use actual data traces of the group membership in an MBONE conference to gain insight into the transient changes of the load at each node. In a subsequent companion technical report [19] we will present a set of soft-state protocol mechanisms that maintain the hypercube topology without requiring any entity to have global state information.

Key Words: *Large Scale Multicast Applications (LSMA), IP Multicast, Multicast Communications, Implosion Problem, Hypercubes.*

*This work is supported in part by a National Science Foundation CAREER Grant (NCR-9624106).

1 Introduction

Recently emerging large scale multicast applications have increased the need for advanced multicast services on the Internet. These services are implemented on top of the basic connectionless IP Multicast service, which does not guarantee reliable or in-sequence delivery [7].

In the basic multicast service, a user joins a multicast group simply by indicating interest in receiving data sent to that group. Any packet that is transmitted to a multicast group is forwarded to all members of the group. Mechanisms for error control, rate control, or in-sequence delivery are not part of the basic service. To implement these mechanisms, multicast group members must exchange control information with each other. However, the exchange of control information in large multicast groups creates scalability problems.

Consider, for example, the implementation of a reliable multicast service. A unicast protocol with a single sender and a single receiver requires the receiver to send positive (ACKs) or negative acknowledgment packets (NACKs) to the sender to indicate reception or loss of data. If the same mechanisms are applied to large groups, the sender would soon be flooded by the number of incoming ACKs or NACKs; this is referred to as the *ACK Implosion Problem* [6, 13].

In recent years, many techniques and protocol mechanisms have been proposed to solve the ACK implosion problem [2, 5, 6, 9, 10, 12, 13, 16, 17, 18, 21, 22, 23, 25, 27, 28, 31, 32, 34], mostly in the context of providing a reliable multicast service. Early proposals were targeted at local area networks and exploited the broadcast capabilities in such networks [25, 27]. In packet-switching networks, we find two approaches to deal with the volume of control traffic. In one approach, control information is broadcast to all members of the multicast group; the volume of control traffic is limited based on the volume of data traffic or on the size of the multicast groups [3, 9, 26]. The drawback of this approach is that the each multicast member has to reduce its control traffic as the multicast group grows. In the second approach, the group members are organized in a logical graph, henceforth called *control topology*. Control information can only be exchanged between group members which are neighbors in the logical graph. By merging control information received from their neighbors, the dissemination of control information can be made efficient. Control topologies that have been considered in the literature include rings [5, 31] and trees [12, 17, 22, 34]. In Figure 1 we illustrate the relationship between the network topology, basic multicast service, and control topologies.

This paper proposes a new approach for disseminating control information between the members of a multicast groups. We propose to organize group members as nodes of an *n-dimensional hypercube*, and disseminate control information in trees that are embedded in the hypercube. We claim that the hypercube has excellent scalability properties, making it the preferred choice for multicast applications with very large group sizes. Our key contributions are as follows. We show how to construct a hypercube control topology with simple boolean operations and present an algorithm to embed rooted spanning trees into the hypercube.

Further, we show that control topologies that are using trees do not balance the load for processing control information well among the multicast group members. We believe that

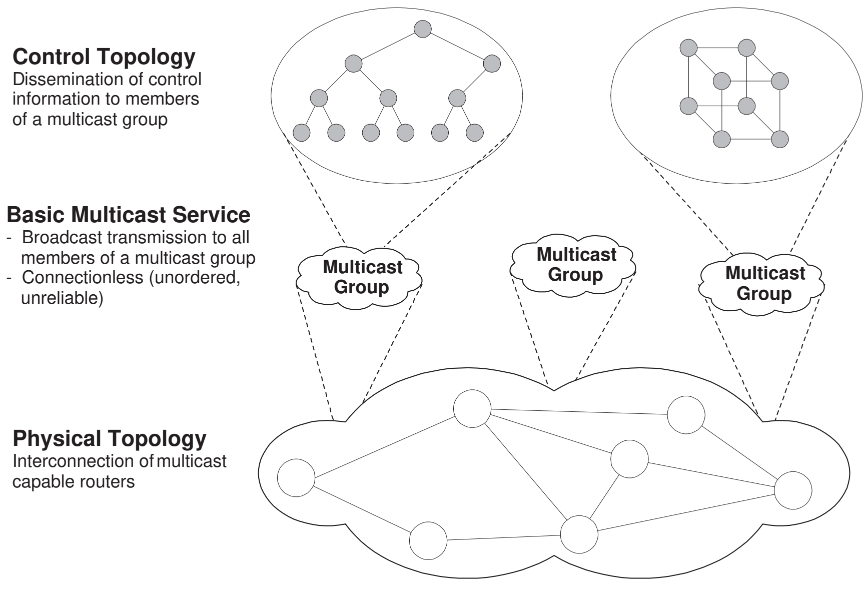


Figure 1: Multicast Framework.

this data point is important, as recent research on protocol support for scalable multicast applications is focusing on tree-based control topologies [12, 17, 22, 34]. We show that the hypercube is superior when it comes to balancing the load for processing control information among multicast group members. Since poor load-balancing results in bottlenecks in the control topology, better load-balancing improves the scalability to large multicast groups.

In this paper we do not address the relationship between multicast routing and the control topology. While a coupling of routing and control topology improve the performance of a multicast protocol, we believe that determining how such a coupling can be attained for the hypercube is beyond the scope of this paper. Our proposed hypercube control topology and the presented quantitative results are orthogonal to the degree to which routing and control topology are coupled.

The remainder of the paper is structured as follows. In Section 2 we review the existing proposals for disseminating control information to the members of a multicast group. In Section 3 we present the hypercube as a new solution to disseminate control information in a multicast group. In Section 4 we analyze the scalability properties of a hypercube and compare them with those of other control topologies. In Section 5 we present an empirical evaluation of various control topologies using actual traces from a large multicast session on the MBONE. In Section 6 we present our conclusions and discuss future work.

2 Control Topologies for Multicast Communications

In this section we review currently used control topologies for disseminating control information in multicast groups, focusing on the ability of these topologies to support large multicast groups. We assume the existence of a higher-layer protocol mechanism that separates data and control. The hypercube is used only for transmitting control information; data is transmitted using the basic IP multicast service. Throughout this paper we assume that many, possibly all, multicast group members are transmitters of information. We assume that routers or network caches are not involved in processing control information.

It is convenient to view the members of a multicast group as a set of nodes V. Nodes are numbered in an arbitrary sequence, that is $V = \{1, 2, \ldots, N\}$. We assume that each node can directly communicate with any other member of the group.

2.1 No Control Topology

Several protocols that extend the basic IP Multicast service do not provide a topology for disseminating control information. Instead, the control information from any node is broadcast to all other nodes in the group. Such protocols must restrict the volume of control information, since otherwise the scalability is severely impeded. The RTP protocol [26] limits the total amount of control traffic to 5% of the data traffic. In [3], feedback from receivers to the senders is adapted to the size of the multicast group.

Among reliable multicast protocols, a technique known as *NACK suppression* [9, 25] or *damping* [27] is used to contain control traffic without a control topology. Here, a multicast group member with a control packet to send is forced to queue this packet for a random time interval before it can be transmitted. If a member receives a control packet which matches a queued packet, it cancels the transmission. For large group sizes, however, the random queueing time must be set to a large value, resulting in slow feedback times for the control information.

Yet another set of protocols without a control topology employ a central controlling station which coordinates ordering and reliability [4, 8, 10]. Due to the load at the controlling station, the scalability of such protocols is limited.

2.2 Ring Topology

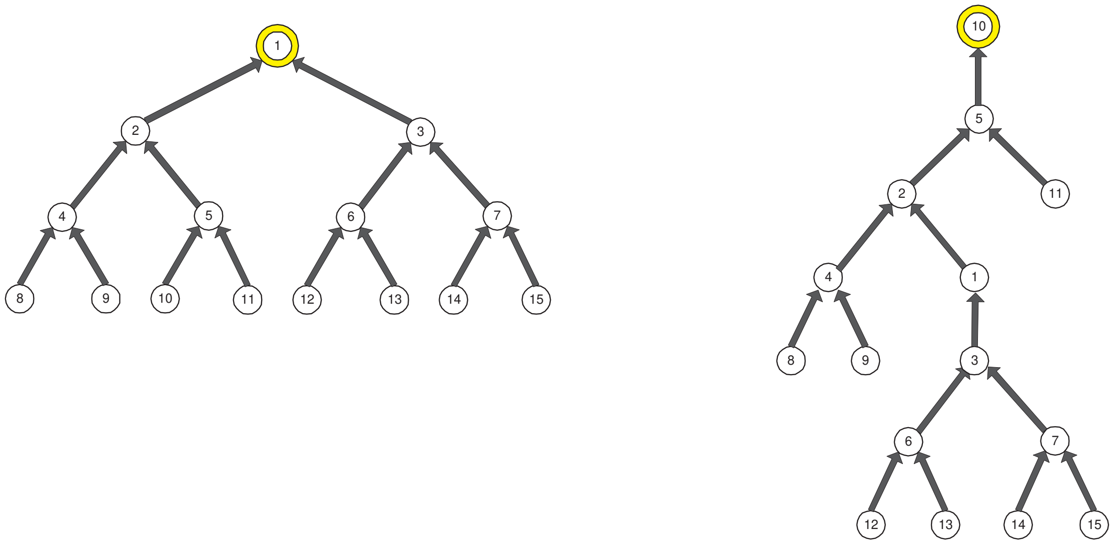
Ring control topologies have been implemented to provide a reliable multicast service with total ordering of messages [5, 31]. In these protocols, the multicast group is structured as a logical ring, and a token is passed around the ring. Control messages are unicast between the current holder of the token and other nodes. The scalability of ring topologies is only moderate since control messages are always directed to the token holder, thus creating a bottleneck at that node. Also, the time to pass the token around the ring increases with the group size, resulting in decreased overall throughput.

2.3 Tree Topology

Tree topologies assume that the members of a multicast group are organized as a *rooted spanning tree*, and all control information is transmitted along the edges of a tree. For each multicast group member, there is one spanning tree where this member is the root of the tree. We use T_k to denote the spanning tree with node k as root. Node l transmits a control message to the root node k by passing the message to its immediate ancestor in T_k, the tree rooted at k. Tree topologies achieve scalability by exploiting the hierarchical structure of a tree. For example, by 'merging' acknowledgments at the internal nodes of a tree, the number of acknowledgments received by a group member is limited to the number of children; thus, ACK implosion is avoided. A drawback of tree-based topologies is the overhead in constructing and maintaining the tree. Note that the tree must be dynamically modified both in response to host failures and to members joining and leaving the tree.

Several tree-based control topologies have been proposed for transmission of control information [11, 17, 22, 25, 34], mostly for multicast groups with only a single sender. We discuss the K-ary shared tree [17] or Pivot tree [18].

The shared tree topology is derived from a single balanced K-ary tree with root r. If some other node $k \neq r$ becomes the root, then the tree is *re-hung* with node k as new root [17]. Re-hanging trees with a new root is illustrated in Figure 2. In Figure 2(a) we show a binary (2-ary) tree with node 1 as root. Figure 2(b) depicts the same tree, 're-hung' for node 10 as root node. Re-hanging a tree does not increase the number of children of each node, however, after re-hanging, the tree may no longer be balanced. Note that the longest path to the root in the re-hung tree in Figure 2(b) is twice as long as in the original tree in Figure 2(a).



(a) Original Tree Rooted at Node 1. (b) Re-hung Tree Rooted at Node 10.

Figure 2: Shared Binary Tree.

Among the currently considered topologies, tree-based topologies seem to be most suited to support large multicast groups. However, the trees created by re-hanging a shared tree burden some nodes with an unproportionally high load for processing control information. In the next section we propose a new control topology, derived from a hypercube, that offers better load balancing, and as a result, has better scalability properties than tree-based solutions.

3 The Hypercube Control Topology

In this section we propose the hypercube as a new control topology for multicast communications. We propose to organize the members of a multicast group as the nodes of a logical n-dimensional hypercube. We present a method for embedding spanning trees into the hypercube and use these spanning trees for disseminating control information.

3.1 Hypercube and Tree Embeddings

An *n-dimensional hypercube* is a graph with $N = 2^n$ nodes where each node is labeled by a bit string $k_n \ldots k_1$ ($k_i \in \{0, 1\}$). Nodes in the hypercube are connected by an edge if their bit strings differ in exactly one position. In Figure 3 we depict hypercubes of dimensions $n = 1$ to $n = 4$.

Hypercubes have been studied extensively by the parallel computing community; they are deemed attractive as a multiprocessor architecture because of their symmetry, the short distances between nodes, and the number of alternative routes. The literature on hypercubes is rich, and we refer to [14, 15, 24] as excellent sources on the topic.

We propose to organize the members of a multicast group as the nodes of a hypercube. Then we embed spanning trees into the hypercube and disseminate control information along the edges of the spanning trees. The advantage of using trees embedded into the hypercube over the re-hung shared trees [17] will become clear in Section 4 where we analyze the performance of control topologies.

Past research on parallel algorithms has produced numerous algorithms for embedding trees in hypercubes (see [15] for an overview). The goal of these these algorithms is to assign a parallel computation, represented as a tree, into a multicomputer with an hypercube interconnection network. These algorithms make a number of assumptions that are not applicable in the context of a multicast group. First, most algorithms assume a static hypercube. Second, with few exceptions [29, 30], these algorithms assume a complete hypercube, i.e., $N = 2^n$. Both assumptions are not realistic for multicast groups with a dynamically changing membership.

For multicast communications, we have to consider *incomplete* hypercubes with $N < 2^n$ nodes. This results in a requirement that the embedded spanning trees in an incomplete hypercube with node set V only contain nodes in V. We refer to such trees as *completely contained* in the incomplete hypercube. Furthermore, we have to consider that the multicast group membership changes dynamically. Since adding and removing nodes may degenerate the compact structure of a hypercube, we need to have mechanisms in place that keep the dimension of the hypercube as small as possible; we refer to this property as *compactness*.

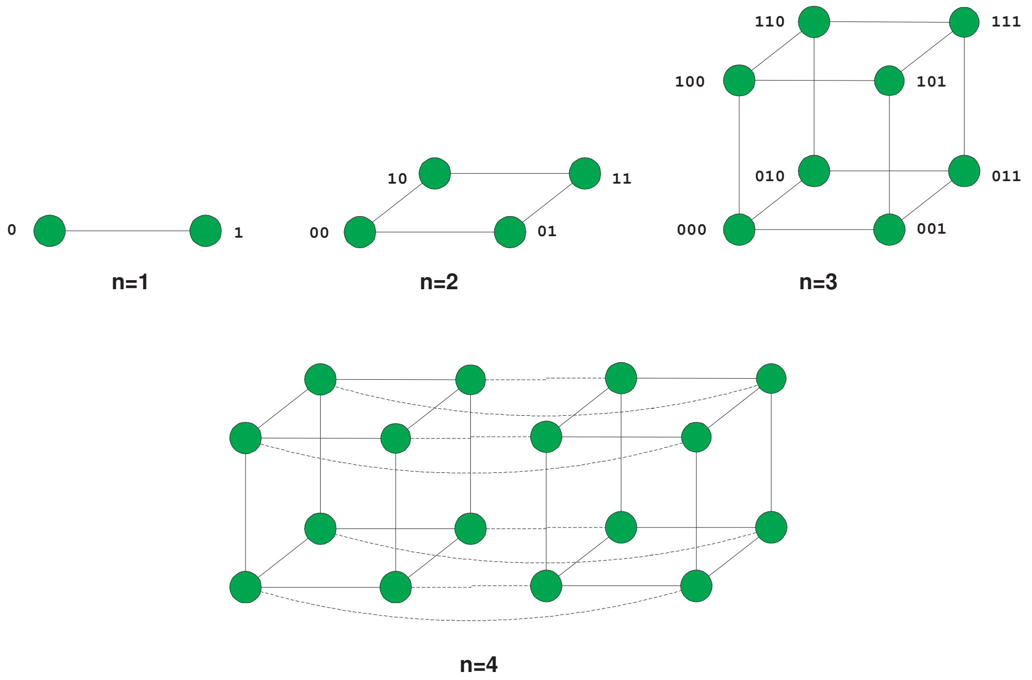


Figure 3: n-dimensional Hypercubes.

3.2 Gray Ordering of Hypercube Nodes

The key to ensure complete containment of all spanning trees and maintain compactness of the incomplete hypercube is the selection of an appropriate ordering of the nodes.

The standard ordering of hypercube nodes interprets the label of a node as a binary number. Specifically, the number $a = \sum_{i=1}^{n} a_i \cdot 2^{i-1}$ is associated with the node labeled $Bin(a) := a_n \dots a_1$ ($a_i \in \{0, 1\}$). With the ordering imposed by the numbers, compactness can be achieved by ensuring that in a multicast group with N members the positions $Bin(0), Bin(1), \dots, Bin(N-1)$ are always occupied. However, using this ordering there it is not clear how to construct spanning trees that satisfy the complete containment condition.

We propose to use a different ordering of the nodes, which is based on an ordering obtained by interpreting node labels using *Gray codes*. A Gray code, denoted by '$G(\cdot)$', is defined by the following properties [24]:

- The values are unique. That is, if $G(i) = G(j) \Rightarrow i = j$.
- $G(i)$ and $G(i+1)$ differ in only one bit, for $0 \leq i < 2^{d-1} - 1$.
- $G(2^{d-1} - 1)$ and $G(0)$ differ in only one bit.

In other words, a Gray code corresponds to a Hamiltonian walk on the hypercube [15].

Let $Bin(i)$ denote the label of the ith node in the binary ordering, it is easy to verify that the following generates a Gray code:

$$G(i) := Bin(i) \otimes Bin(i/2)$$

where '$\otimes$' is the XOR operator and '$x/2$' is an integer division by 2. We use $G^{-1}(\cdot)$ to denote the inverse of $G(\cdot)$, that is, $G^{-1}(G(i)) = i$.

Clearly, with this ordering compactness can be enforced by ensuring that the members of a hypercube with K nodes occupy positions $G(0), G(1), \ldots, G(N-1)$.

Example: Consider the ordering of nodes in a 3-dimensional hypercube. Refer to Figure 3 for the labeling of nodes. In the following table, we show the position number i, the binary interpretation $Bin(i)$, and the Gray code $G(i)$:

Position Number i	0	1	2	3	4	5	6	7
$Bin(i)$	000	001	010	011	100	101	110	111
$G(i)$	000	001	011	010	110	111	101	100

Thus, using the standard ordering $Bin(i)$ a multicast group with $K = 5$ members would occupy the following positions in the hypercube: 000, 001, 010, 011, 100. In contrast, using a Gray code $G(i)$ occupies the following positions: 000, 001, 011, 010, 110.

3.3 Tree Embedding in Gray-ordered Hypercubes

We now present an algorithm to embed spanning trees into hypercubes that use Gray codes for ordering the nodes. The embedding of spanning trees in the hypercube is calculated locally: A node with label $G(x)$ directly obtains the address of its parent node in the tree with root $G(r)$. The ability to calculate the embedded tree in a distributed fashion will be exploited in [19] where we present protocol mechanisms to maintain a hypercube topology.

For a Gray-ordered hypercube which preserves compactness as shown in the previous subsection, we present an algorithm that always generates a *completely contained* spanning tree. The algorithm is presented in Figure 4 and consists of a single procedure '*Parent*'. Given two node labels I and R, the procedure computes the label of the parent of node I in the spanning tree with R. The procedure *Parent* simply flips a single bit in label I. If I is smaller than R in the Gray ordering the parent of I is obtained by flipping the least significant bit in I where I and R differ. Otherwise, the procedure flips the most signficiant bit in I where I and R differ. If each node performs the procedure *Parent* for a root node R, we obtain a spanning tree with root R embedded into the hypercube.

In Figures 5 and 6 we show the embeddings of the spanning trees in a 3-dimensional hypercube for root nodes 1 and 5, respectively. Intentionally, we have depicted an incomplete hypercube with $N = 7 < 2^3$ nodes.

All trees that are constructed by procedure *'Parent'* have the following set of properties. The properties follow directly from the procedure and are shown without proof.

> **Property 1:** A node and its parent always have a Hamming distance of 1.
>
> **Property 2:** The path length between a node and a root is given by their Hamming distance.

Input: Label of the i-th node in the Gray encoding:
$G(i) := I = I_n \dots I_2 I_1$,
and the label of the r-th node ($\neq i$) in the Gray encoding:
$G(r) := R = R_n \dots R_2 R_1$.
Output: Label of the parent node of node I in the embedded tree rooted at R.

Procedure Parent (I, R)
    If $(G^{-1}(I) < G^{-1}(R))$
        // Flip the *least significant bit* where I and R differ.
        Parent $:= I_n I_{n-1} \dots I_{k+1} (1 - I_k) I_{k-1} \dots I_2 I_1$
            with $k = \min_i (I_i \neq R_i)$.
    Else // $(G^{-1}(I) > G^{-1}(R))$
        // Flip the *most significant bit* where I and R differ.
        Parent $:= I_n I_{n-1} \dots I_{k+1} (1 - I_k) I_{k-1} \dots I_2 I_1$
            with $k = \max_i (I_i \neq R_i)$.
    Endif

Figure 4: Tree Embedding Algorithm.

Property 3: In a hypercube with N nodes, all trees have a depth of $\lceil \log_2(N) \rceil$. If $N = 2^n$, the embedding results in a binomial tree.[1]

Property 4: If *Parent* (I, R) is the p-th node in the Gray encoding, then the following holds: $p \leq \max\{i, r\}$.

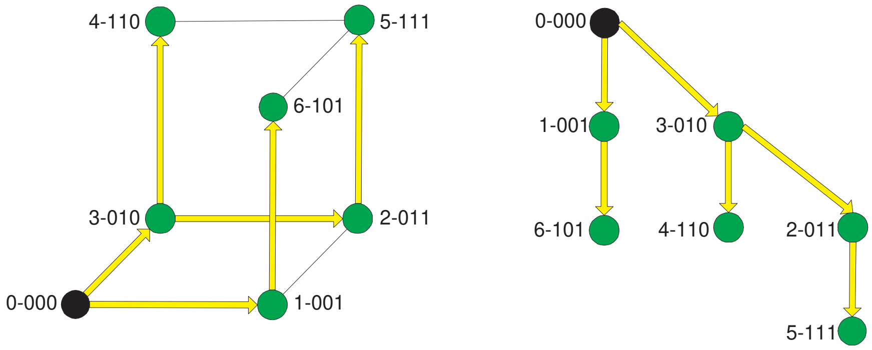
Property 4 of the algorithm guarantees complete containment of all embedded trees. Next we analyze the properties of the proposed hypercube control topology and compare it against tree-based solutions.

4 Analysis of Scalability Properties

To gain insight into the scalability property of the hypercube control topology, we conduct a performance comparison of the hypercube with the K-ary shared tree discussed in Subsection 2.3. We derive a set of performance measures that are computed as averages over rooted spanning trees. For each node, there is one spanning tree where this node is the root. The spanning trees with some node 'x' as root are obtained as follows:

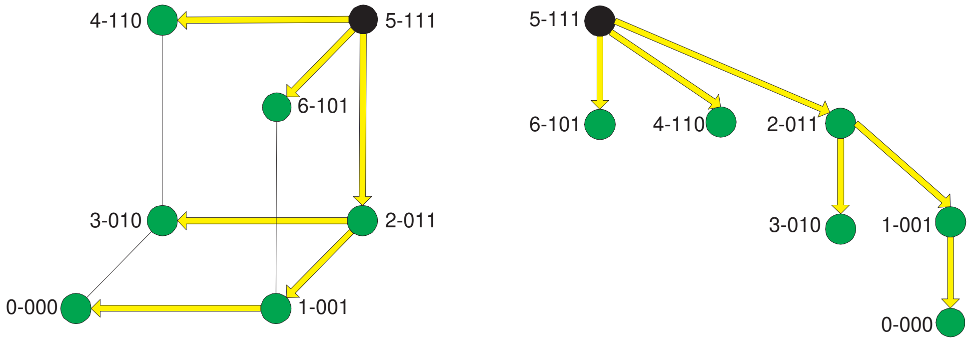
- **Hypercube:** A tree with node x as root is embedded in an n-dimensional hypercube using the Algorithm in Figure 4.
- **Shared K-ary Tree:** A given K-ary tree, the 'original tree' (see Section 2), is re-hung with node x as root.

[1] A *binomial tree* of height 0 is a single node. For all $i > 0$, a binomial tree of height i is a tree formed by connecting the roots of two binomial trees of height $i - 1$ with an edge and designating one of these roots to be the root of the new tree (cited from [24]).



(a) Embedding (b) ResultingTree

Figure 5: Embedding a Tree with Node 1 as Root.



(a) Embedding (b) ResultingTree

Figure 6: Embedding a Tree with Node 5 as Root.

In our analysis we assume that that communication within a multicast group is *symmetric*, i.e., on the average, each member of the group generates the same amount of traffic. In contrast, an asymmetric groups is characterized by few members of the group generating most or all of the traffic.[2]

The scope of our investigation is limited to quantitative aspects of disseminating control information. We do not consider the effects of protocol processing or routing issues. Furthermore, we consider generic transmission of control messages without assumptions on a particular control function (flow control, error control, etc.) as in [17, 20, 23, 33].

4.1 Performance Measures

We define a set of performance measures which reflect the load for processing control information at a multicast group member. In all configurations considered, control information is transmitted along the edges of a *rooted spanning tree*. Recall that we use T_l to denote the spanning tree with node $l \in V$ as root.

[2]A multicast web server is an extreme example of an asymmetric group; here, the server is the only member of the group that generates traffic.

For most control function, a good indicator for the load at a node in a control tree is the number of its children. We define:

$$w_k(T_l) := \text{ Number of children of node } k \in V \text{ in tree } T_l.$$

Since control functions may incur a load at a node that is proportional to the size of the subtree rooted as this node, we define:

$$\begin{aligned} v_k(T_l) := \ & \text{Number of descendants of node } k \in V \text{ in tree } T_l \text{ (including node } k\text{),} \\ & \text{where the } \textit{descendants} \text{ of node } k \text{ in tree } T_l \text{ are the nodes that} \\ & \text{have node } k \text{ on their path to root } l. \end{aligned}$$

Finally, since the path length in a control tree indicates the delay of passing control information in the tree, we define a measure that expresses this delay:

$$p_k(T_l) := \text{ Length of the path from node } k \text{ to root node } l \text{ in } T_l.$$

We define more concise measures by taking the average over all trees T_l with $l \in V$, denoted as w_k, v_k, and p_k. These measures are defined as follows:

$w_k := \frac{1}{N}\sum_{l=1}^{N} w_k(T_l)$	Average number of direct children of node $k \in V$ in a spanning tree.
$v_k := \frac{1}{N}\sum_{l=1}^{N} v_k(T_l)$	Average size of the subtree of node $k \in V$.
$p_k := \frac{1}{N}\sum_{l=1}^{N} p_k(T_l)$	Average path length from node k to the root.

To further condense the amount of data, we take the averages and maxima of the above values and obtain:

$$w_{avg} := \frac{1}{N}\sum_{k=1}^{N} w_k \qquad v_{avg} := \frac{1}{N}\sum_{k=1}^{N} v_k \qquad p_{avg} := \frac{1}{N}\sum_{k=1}^{N} p_k$$

$$w_{max} := \max_k w_k \qquad v_{max} := \max_k v_k \qquad p_{max} := \max_k p_k$$

We use the values of w_{max}, v_{max}, and p_{max} to calculate measures for the degree of load balancing of a topology. Specifically, we use the ratios w_{max}/w_{avg}, v_{max}/v_{avg}, and p_{max}/p_{avg} to compare the worst-case node and average node for a control topology. Our expectation is that a control topology with good scalability properties must balance the load incurred at a node, which is reflected in the need for low maximum-to-average ratios.

In the following we present the results of the above measures for the K-ary tree and the hypercube. The complete set of derivations are presented in the appendix of this paper.

4.2 Analysis of the n-dimensional Hypercube

The performance measures for the hypercube can be calculated by taking advantage of the highly symmetric topology and the properties of the Gray code. In Appendix A we show the complete set of derivations. For a full hypercube, that is, $N = 2^n$, we obtain the following exact formulas:

$$\begin{aligned}
w_{avg} &= 1 - \frac{1}{N} \\
w_{max} &= 2 - \frac{\log_2 N + 2}{N} \\
v_{avg} &= \frac{1}{2}\log_2 N + 1 \\
v_{max} &= \frac{1}{8}(\log_2 N)^2 + \frac{3}{8}\log_2 N + 1 \\
p_{avg} &= \frac{1}{2}\log_2 N \\
p_{max} &= \frac{1}{2}\log_2 N
\end{aligned}$$

We have also derived bounds for these performance measures in the incomplete hypercube, that is, $N < 2^n$. While these bounds, derived in Appendix B, are not sharp, they exhibit the same growth behavior as the exact expressions for the full hypercube. Therefore, we will deal only with the exact expression in our numerical examples.

4.3 Analysis of the Shared K-ary Tree

Recall that control trees in the shared K-ary tree are obtained from a single tree by re-hanging this tree with different nodes as root [17, 18]. In Figure 2 we showed an example of re-hanging a binary tree.

As K is increased, the maximum path length from a node to the (re-hung) root is decreased. This, however, increases the load on the node that is the root in the original tree. In the extreme case, we have $N = K$ and obtain a *star topology* where re-hanging the tree always results in $N - 1$ nodes hanging off the original root of a star topology.

Let us assume for simplicity that all leaves of the tree are occupied, i.e., there exists a $d \geq 0$ with $N = \sum_{l=0}^{d} K^l$. Then we obtain:

$$\begin{aligned}
w_{avg} &= 1 - \frac{1}{N} \\
w_{max} &= K + \frac{1}{N} \\
v_{avg} &= 2d + \frac{K-5}{K-1} + \frac{6d}{N(K-1)} + \frac{4(K-2)}{N(K-1)^2} + \frac{4(d+1)}{N^2(K-1)^2} \\
v_{max} &= \begin{cases} \dfrac{5N}{8} + \dfrac{1}{4} - \dfrac{11}{8N} & \text{if } K = 2 \\[2ex] \dfrac{(K-1)N}{K} + \dfrac{2}{K} - \dfrac{1}{KN} & \text{otherwise} \end{cases}
\end{aligned}$$

$$
\begin{aligned}
p_{avg} &= 2d - \frac{4}{K-1} + \frac{6d}{N(K-1)} + \frac{4(K-2)}{N(K-1)^2} + \frac{4(d+1)}{N^2(K-1)^2} \\
p_{max} &= 2d - \frac{3}{K-1} + \frac{3(d+1)}{N(K-1)}
\end{aligned}
$$

The derivations for these performance measures are given in Appendix C.

4.4 Discussion

We now use the derived measures to demonstrate how the shared tree and hypercube control topologies scale when the size of a multicast group grows large.

Let us first examine the number of children of a node, w_{avg} and w_{max}. Since, for all spanning trees, the average number of children is $w_{avg} < 1$, i.e., on the average a node in a control tree has only one child, we only compare the ratios w_{max}/w_{avg}:

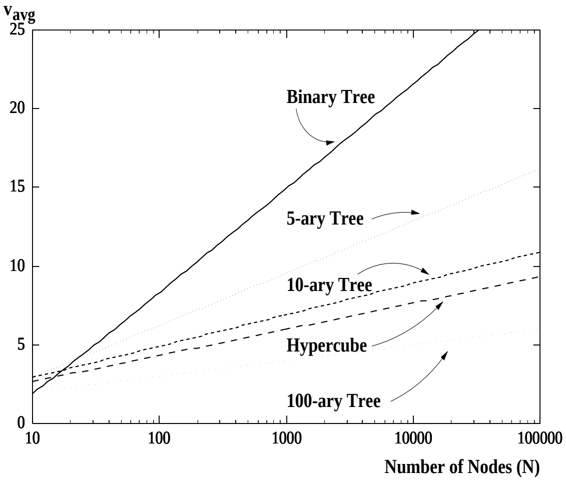
K-ary Tree:	$w_{max}/w_{avg} \longrightarrow_{N\to\infty} K$
Hypercube:	$w_{max}/w_{avg} \longrightarrow_{N\to\infty} 2$

Thus, in a K-ary tree there is a node that has, on the average, K times as many children as the average node. The hypercube, in contrast, is better load-balanced. Here, the difference between the worst-case and the average case is only a factor of 2. (In the K-ary tree, the maximum is attained for the root in the original K-ary tree. The hypercube attains the maximum at the node with label `00...0`).

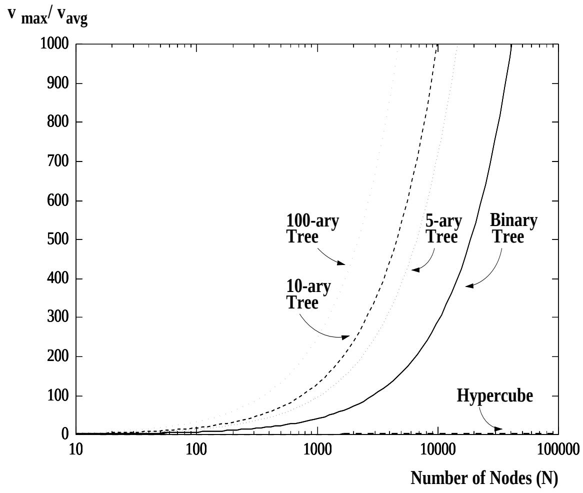
In Figure 7(a) we present a graph where we plot the values for v_{avg} by varying the number of nodes N. We present results for the shared K-ary tree (with $K = 2, 5, 10, 100$) and the hypercube. In the figure we see that, in a K-ary shared tree, v_{avg} decreases for increased values of K. Note that, over the entire range of values, v_{avg} for the hypercube is smaller for the 10-ary tree.

The comparison of the ratios v_{max}/v_{avg}, depicted in Figures 7(b), reveals a problem with load balancing for the shared tree topologies. Since v_{max} increases linearly in N, all shared trees have a bottleneck at the node were the maximum is attained. For large values of K, scalability problems arise even for small group sizes. For all trees, the maximum-to-average ratio exceeds 100 when the number of nodes has more than a thousand nodes. In a direct comparison with the K-ary tree, the value of v_{max}/v_{avg} for the hypercube appears almost insignificant.

In Figures 8(a) and 8(b) we present the results for the path lengths. It is interesting to note that the average path to the root is shorter in the hypercube than in a 10-ary tree. For the entire range of values shown, the embedded trees from the hypercube have less than 10 children per node, resulting in longer path length. However, due to re-hanging the shared 10-ary tree will result – on the average – in longer paths length to the root. Figure 8(b) shows that load balancing is not an issue when considering the path lengths. As the size of the multicast group is increased, the ratio p_{max}/p_{avg} quickly approaches 1 in all topologies.

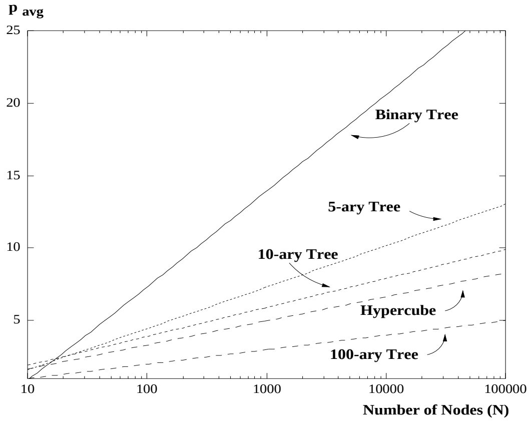


(a) Results for v_{avg}.

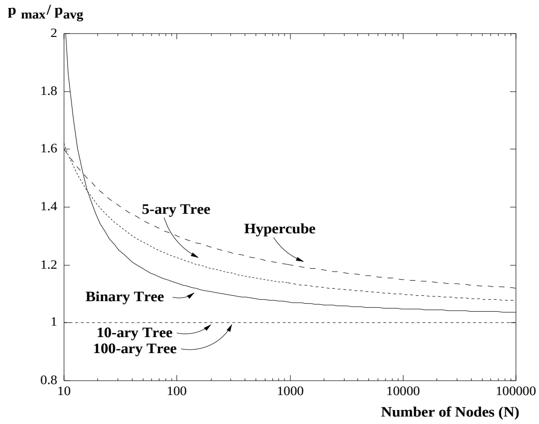


(b) Results for v_{max} / v_{avg}.

Figure 7: Comparison of Average Number of Descendants.



(a) Results for p_{avg}.



(b) Results for p_{max}/p_{avg}.

Figure 8: Comparison of the Average Path Lengths.

In summary, the hypercube appears very suitable to support large multicast groups. A comparison with shared K-ary tree showed that the hypercube has all the advantages, and none of the disadvantages of the shared K-ary tree. Particularly, the hypercube provides an excellent balance of the average and worst-case load at the nodes. The load-balancing indicators w_{max}/w_{avg} and v_{max}/v_{avg} clearly demonstrate that the shared tree topology has problems when scaled to very large group sizes. Figure 7(b) illustrates that the load-balancing problems of the shared K-ary seem to be independent of the selection of K. In contrast, the hypercube topology does not have these scalability problems.

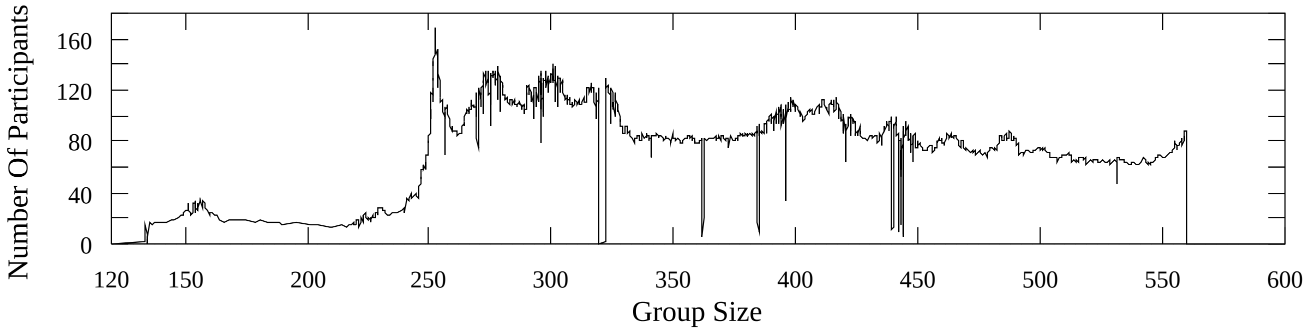


Figure 9: Group Size of the MBONE Session.

5 Empirical Evaluation of Dynamic Behavior

In this section, we present an empirical evaluation of the hypercube control topology. We measure the dynamic behavior of the hypercube control topology, and compare the results with the K-ary shared tree topology.

The basis for our empirical evaluation are data traces of an MBONE session obtained with the *Mlisten* tool [1]. The authors of [1] have made available detailed traces of the multicast group membership of MBONE sessions. The traces that are the basis for our evaluation are the video transmission sessions for the *NASA's STS-80C space shuttle mission* measured over a period of 27 days (657 hours) from November 8, 1996 to December 4, 1996.

The varying size of the group membership over the measurement period from 120 hours to 600 hours is shown in Figure 9. Over the entire length of the experiment, the MBONE sessions had 1,874 different hosts participating with a maximum of 169 simultaneous users (at $t = 252$ hours into the experiment). Note in Figure 9 the occasional drops in the group size, e.g., at $t = 319$ hours. Even though these drops are likely due to artifacts of the measurement experiments, we assume that Figure 9 reflects the actual group size.

We use this trace to track the dynamic behavior of the performance measures from Subsection 4.1 which indicate the average load and the degree of load balancing.

As before, we perform a comparison between the hypercube topology and the shared K-ary tree. In all topologies, we assume that, whenever the group membership has changed, the resulting topology is as compact as possible. In tree topologies, we keep the tree depth minimal, and in the hypercube topology, we attempt to minimize the hypercube dimension. For a hypercube this means we assume that compactness is always maintained. For the tree, our scheme ensures that all nodes at level 0 (root) to d are filled before filling any node at level $d + 1$. This assumption ensures that we always present the most optimistic results for the shared K-ary tree. Actual algorithms for restructuring shared K-ary trees, e.g., [17], may yield worse results.

Next we discuss the outcome of the experiment:

- **Number of children of a node**: In Figure 10 we depict the ratio $w_{max}/\overline{w}$ which reflects the skewness of load balancing.[3] Note that the ratio is lowest for the hypercube. For K-ary trees, the ratio increases as the maximum number of children of a node is increased.

• **Number of descendants of a node:** In Figure 11 we depict the values for the ratio $v_{max}/\overline{v}$. (The values of $\overline{v}$ are ≤ 4 for the hypercube, ≤ 10 for the binary tree, ≤ 6 for the 5-ary tree, and ≤ 3 for the Star network). The hypercube clearly emerges as the topology with the best load balancing properties.

• **Path Length in Control Topology:** Since for all topologies considered, the maximum value p_{max} is within a constant from the average $\overline{p}$ (see Section 4), we only show the values for $\overline{p}$. In Figure 12 we see that the average path length in the hypercube topology is less than 4 hops, less than that for the 5-ary tree.

6 Conclusions

We have presented a new approach for disseminating control information between the members of a multicast groups. In our approach, we organize the members of a multicast group in a logical hypercube and assign each multicast group member a number that is derived from a Gray code. The Gray encoding enables each node to locally calculate the next hop for transmitting control information. We analyzed the scalability properties of our approach in symmetric multicast groups, i.e, where each group member is a sender. In a comparison with an K-ary shared tree control topology we showed that the hypercube is superior in balancing the load of control information among all nodes.

The findings of this paper have encouraged us to continue our research on using hypercubes for scalable multicast applications. We are currently devising a set of protocol mechanism for maintaining the hypercube topology in a packet-switching network. Shortly, we will present a soft-state protocol that can maintain a hypercube without requiring any network entity to be aware of the global state of the multicast group. The protocol is be self-healing in the sense that it has built-in mechanism for recovering from node or link failures. With the protocol, we can conduct measurement experiments that enable us to do performance comparisons with actual multicast protocols. Also, in future work we will investigate the relationship between routing and control topology, and attempt to trade-off the geographical location (physical address) and node label (logical address) of multicast group members.

7 Acknowledgments

We gratefully acknowledge the help of Prof. Almut Burchard from the Department of Mathematics at Princeton University with devising the tree embedding algorithm. We thank Prof. Mostafa Ammar and Dr. Kevin Almeroth from the College of Computing at Georgia Tech for making available to us the traces of the MBONE measurements.

[3]We mentioned before, that $\overline{w}$ approaches 1 as the number of nodes increases, and is, therefore, not very interesting.

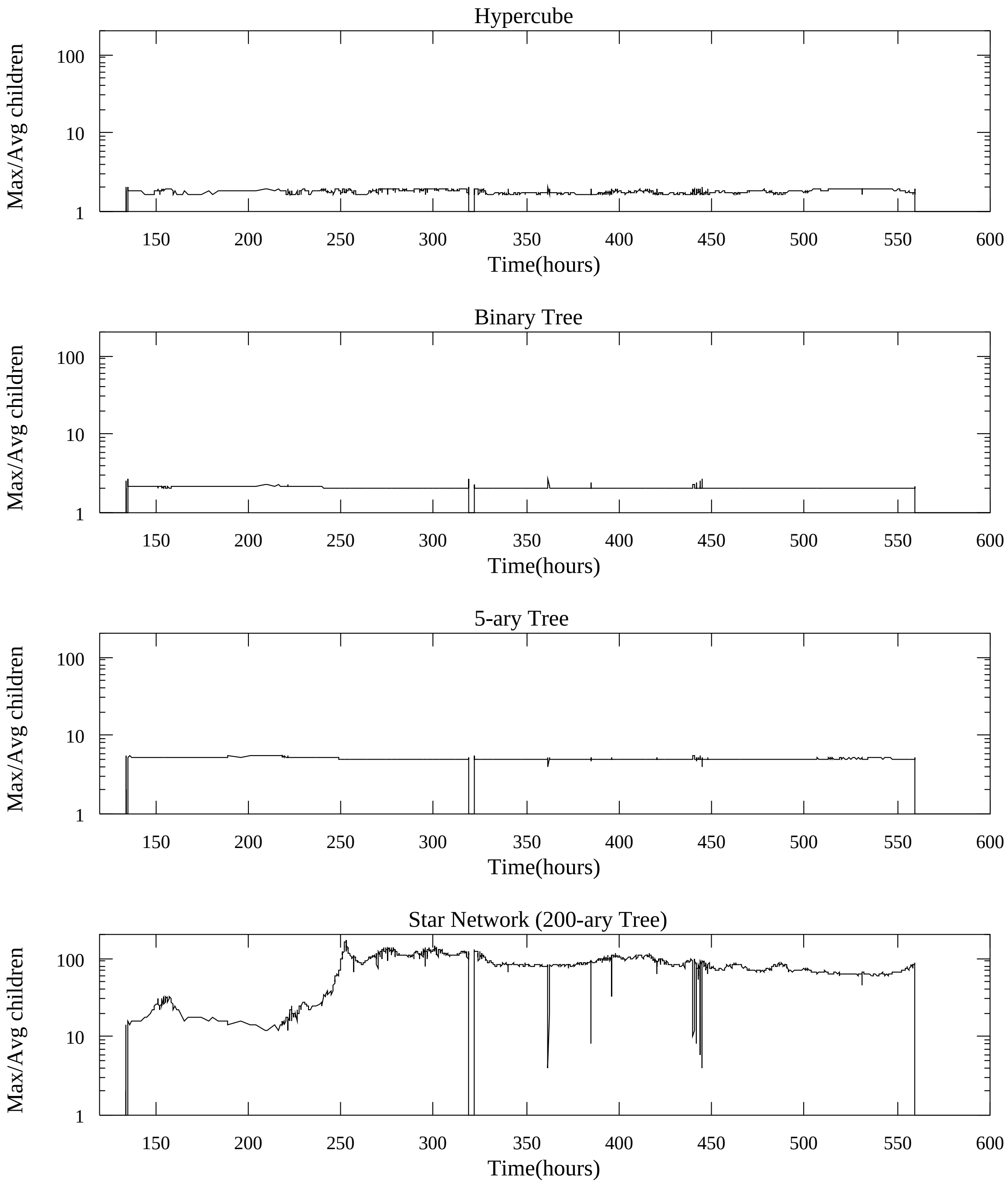


Figure 10: Maximum to Average Children Ratio $w_{max}/\overline{w}$.

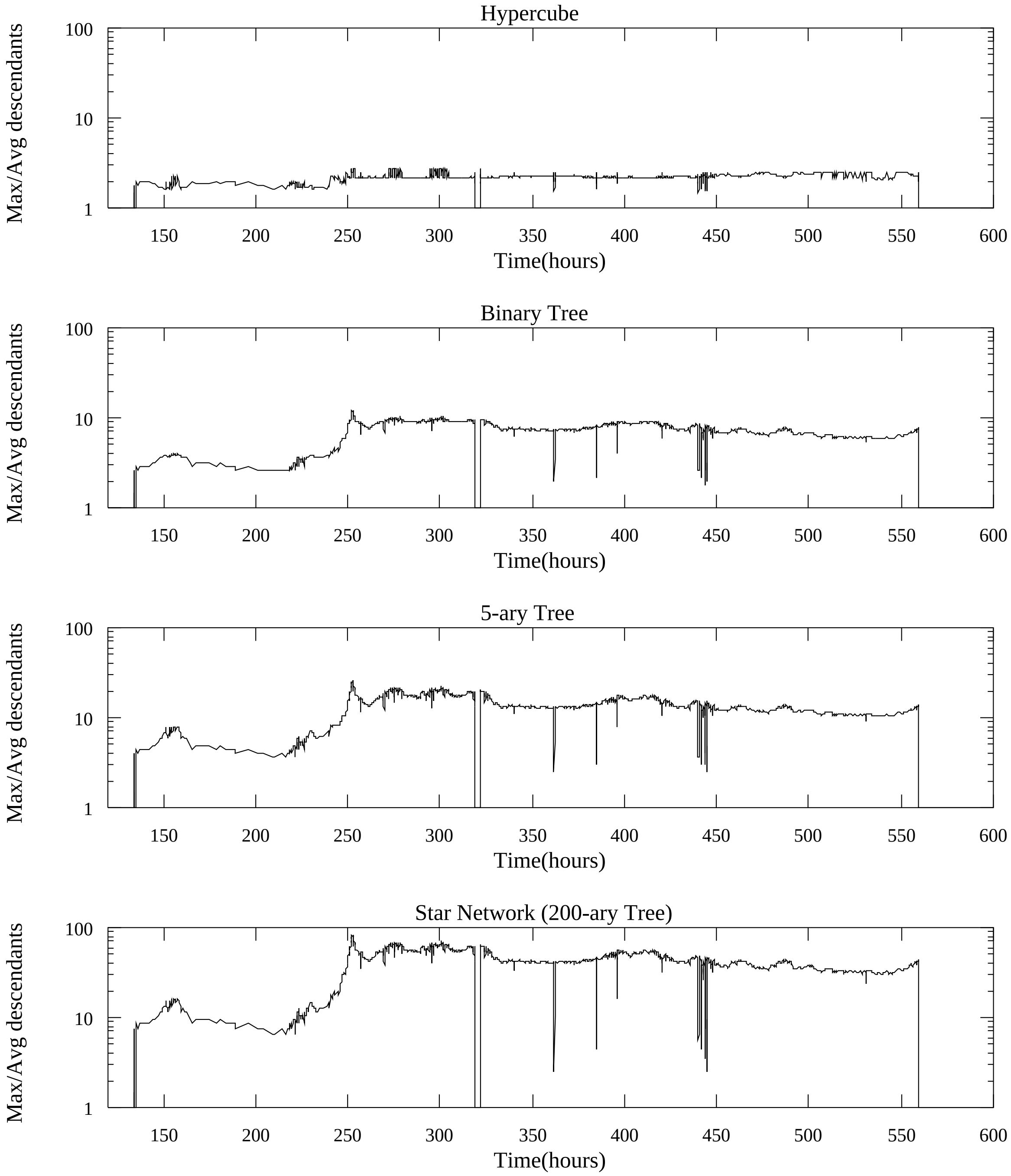


Figure 11: Maximum to Average Descendant Ratio $v_{max}/\overline{v}$.

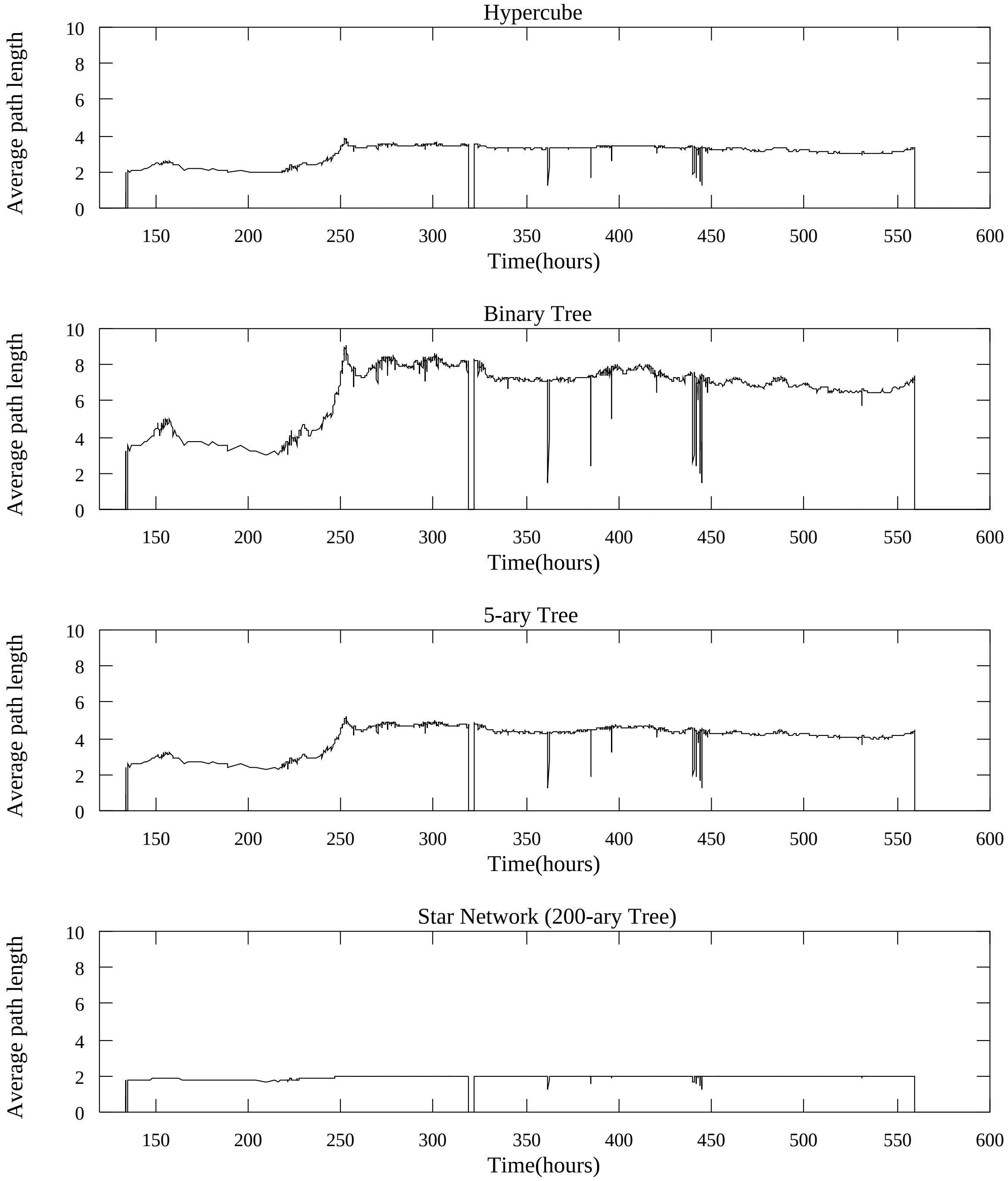


Figure 12: Average Path Length $\overline{p}$.

References

[1] K. Almeroth and M. Ammar. Multicast Group Behavior in the Internet's Multicast Backbone (MBone). *IEEE Communications Magazine*, 35(6), June 1997.

[2] S. Armstrong, A. Freier, and K. Marzullo. Multicast Transport Protocol. Technical Report RFC 1301, Internet Engineering Task Force, February 1992.

[3] J. Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. *Proc. ACM Sigcomm '93*, 23(4):289–298, September 1993.

[4] C. Bormann, J. Ott, H. Gehrcke, T. Kerschat, and N. Seifert. MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport. In *Proc. ICCN '94, San Francisco*, 1994.

[5] J. M. Chang and N. F. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computing Systems*, 2(3):251–273, August 1984.

[6] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proc. ACM Sigcomm '88*, pages 247–256, August 1988.

[7] S. E. Deering and D. R. Cheriton. Host Groups: A Multicast Extension to the Internet Protocol. Technical Report RFC 966, Internet Engineering Task Force, December 1985.

[8] M. F. Kaashoek et. al. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–20, October 1989.

[9] S. Floyd, V. Jacobson, McCanne S, C.-G. Liu, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *Proc. ACM Sigcomm '95*, pages 342–356, August 1995.

[10] A. Frier and K. Marzullo. MTP: An Atomic Multicast Transport Protocol. Technical report, Cornell University, 1990.

[11] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. *ACM Transactions on Computer Systems*, 9(3):242–271, August 1991.

[12] H. W. Holbrook, S. K. Singhal, and D. E. Cheriton. Log-based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proc. ACM Sigcomm '95*, August 1995.

[13] M. G. W. Jones, S. A. Sorensen, and S. Wilbur. Protocol Design for Large Group Multicasting: The Message Distribution Protocol. *Computer Communications*, 14(5):287–297, 1991.

[14] S. Lakshmivarahan and S. K. Dhall. *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems.* McGraw-Hill, New York, 1990.

[15] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufman Publishers, San Mateo, 1992.

[16] B. N. Levine and J.J. Garcia-Luna-Aceves. A Comparison of Known Classes of Reliable Multicast Protocols. In *Proc. IEEE International Conference on network Protocols (ICNP '96)*, 1996.

[17] B. N. Levine, D. B. Lavo, and J.J. Garcia-Luna-Aceves. The Case for Reliable Concurrent Multicasting Using Shared Ack Trees. In *Proc. ACM Multimedia '96*, November 1996.

[18] B. N. Levine and R. Rom. Supporting Reliable Concast with ATM Networks. Technical report, Sun Research Labs SDS-96-0517, January 1997.

[19] J. Liebeherr, T. K. Beam, and B. S. Sethi. HyperCast: A Protocol for Super-Scalable Multicast. Technical report, Polytechnic University, 1998. In Preparation.

[20] J. Nonnenmacher and E. W. Biersack. Performance Modelling of Reliable Multicast Transmission. In *Proc. IEEE Infocom '97*, pages 472–480, April 1997.

[21] C. Papadopoulus, G. Parulkar, and G. Varghese. An Error Control Scheme for large-Scale Multicast Applications. *Submitted for Publicaton*, 1997.

[22] A. Paul, K. K. Sabnani, J. C.-H. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications. Special Issue for Multipoint Communications*, 15(3):407 – 421, April 1997.

[23] S. Pingali, D. Towsley, and J. F. Kurose. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. In *Proc. ACM Sigmetrics '94*, May 1994.

[24] M. J. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill, New York, 2nd edition, 1994.

[25] S. Ramakrishnan and B. N. Jain. A Negative Acknowledgment With Periodic Polling Protocol for Multicast over LANs. In *Proc. IEEE Infocom '87*, pages 502–511, March/April 1987.

[26] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Technical Report RFC 1889, Internet Engineering Task Force, January 1996.

[27] W. T. Strayer, B. D. Dempsey, and A. C. Weaver. *XTP: The Xpress Transfer Protocol.* Addison-Wesley Publishing, 1992.

[28] R. Talpade and M. H. Ammar. Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. In *Proc. 15th IEEE Intl Conf on Distributed Computing Systems (ICDS-15)*, June 1995.

[29] N.-F. Tzeng and H.-L. Chen. Structural and Tree Embedding Aspects of Incomplete Hypercubes. *IEEE Transactions on Computers*, 43(12):1434–1438, December 1994.

[30] N.-F. Tzeng and H. Kumar. Traffic Analysis and Simulation Performance of Incomplete Hypercubes. *IEEE Transactions on Parllel and Distributed Systems*, 7(7):740–754, July 1996.

[31] B. Whetten, S. Kaplan, and T. Montgomery. A High Performance Totally Ordered Multicast Protocol. In *Proc. Infocom '95*, 1995.

[32] X. Rex Xu, A. C. Myers, H. Zhang, and R. Yavatkar. Resilient Multicast Support for Continuous-Media Applications. In *Proceedings of NOSSDAV '97*, 1997.

[33] M. Yamamoto, J. F. Kurose, D. F. Towsley, and H. Ikeda. A Delay Analysis of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. In *Proc. IEEE Infocom '97*, pages 481–479, April 1997.

[34] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proceedings of ACM Multimedia 95*, November 1995.

APPENDIX: Proofs of Scalability Properties

This appendix contains the formal proofs of the scalability properties of the shared K-ary tree and the hypercube as presented in Section 4. We derive the following performance measures for both the shared K-ary tree and the n-dimensional hypercube.

$$w_{avg} \quad w_{max} \quad v_{avg} \quad v_{max} \quad p_{avg} \; p_{max}$$

In Appendix A we show the derivations for a complete hypercube, that is, an n-dimensional hypercube with $N = 2^n$ nodes. In Appendix B we derive bounds for the performance measures for incomplete hypercubes with $N < 2^n$ nodes. In Appendix C, we analyze a complete shared K-ary tree with $N = \sum_{l=0}^{d} K^l$ nodes.

A The n-dimensional Hypercube ($N = 2^n$)

In this section, we prove the properties for trees that are embedded into an n-dimensional hypercube. Recall from Section 3 that all trees embedded into a hypercube with our embedding algorithm are binomial trees. In Figure 13 we illustrate the topology of binomial trees. In the following we assume that the hypercube is complete, i.e., $N = 2^n$.

A.1 Number Of Children

A.1.1 Total Average Number of Children w_{avg}

The quantity w_{avg} is obtained from a general property of trees. Clearly, each node in a tree except the root node is a child node. Thus, the number of children nodes in any tree is given by $N-1$, and the average number of children per node is $(N-1)/N$. Therefore, w_{avg} is obtained as

$$w_{avg} \quad = \quad 1 - \frac{1}{N} \tag{1}$$

A.1.2 Maximum Average Number of Children w_{max}

We claim that the maximum average in a hypercube occurs for the node that is labeled '00..0', and call this node $\underline{0}$. To obtain w_{max} we now calculate the number of children of this node $\underline{0}$ in the embedded tree with root node R. We distinguish three different label types for root node R and perform the calculation independently for each type. The labels are as follows:

Case 1: The label of R contains two or more 1's.

Case 2: The label of R contains a single 1.

Case 3: The label of R is all 0's, i.e., $\underline{0}$ is the root.

We consider each of the cases separately.

Case 1: In this case, the root has j_1 leading zeroes, and j_2 trailing zeroes ($j_1, j_2 \geq 0$). Let 'x' denote either a '0' or a '1', the root R has the following label

$$R = \underbrace{00...00}_{j_1}\ 1\ xx\ldots xx\ 1\ \underbrace{00...00}_{j_2} \tag{2}$$

If S is a child of $\underline{0}$ when R is the root then, by definition of our procedure *Parent()*, the label of S must have one 1 in one of the j_1 leading positions or j_2 trailing positions.

$$\begin{array}{rcl} R &=& \underbrace{00...00}_{j_1}\ 1\ xx\ldots xx\ 1\ \underbrace{00...00}_{j_2} \\ \underline{0} &=& \underbrace{00...00}_{j_1}\ 0\ 00\ldots 00\ 0\ \underbrace{00...00}_{j_2} \\ S &=& \underbrace{00.1.00}_{j_1}\ 0\ 00\ldots 00\ 0\ \underbrace{00...00}_{j_2} \end{array} \quad \text{or} \quad \begin{array}{rcl} R &=& \underbrace{00...00}_{j_1}\ 1\ xx\ldots xx\ 1\ \underbrace{00...00}_{j_2} \\ \underline{0} &=& \underbrace{00...00}_{j_1}\ 0\ 00\ldots 00\ 0\ \underbrace{00...00}_{j_2} \\ S &=& \underbrace{00...00}_{j_1}\ 0\ 00\ldots 00\ 0\ \underbrace{00.1.00}_{j_2} \end{array}$$

Thus, node $\underline{0}$ has $j = j_1 + j_2$ children. For each value of j, there are $(j+1)2^{n-j-2}$ different values of R. (To verify our claim that no node has more children, one can convince oneself that any node $\neq \underline{0}$ cannot have more children in the depicted case.) By multiplying each value with the number of children and summing up the results, we obtain

$$C_1 \ := \ \sum_{j=0}^{n-2} j(j+1)2^{n-j-2} \tag{3}$$

After some algebraic manipulations and with $N = 2^n$ we obtain

$$C_1 \ = \ 2N - n^2 - n - 2 \tag{4}$$

Case 2: In this case the label of the root R only contains a single 1. Assuming that R has j_1 leading 0's, we obtain

$$R = \underbrace{00...00}_{j_1}\ 1\ \underbrace{00...00}_{n-j_1-1} \tag{5}$$

Similar to the previous case, we obtain the following relationship between R, $\underline{0}$, and a child S of $\underline{0}$ in the embedded tree with node R

$$\begin{array}{rcl} R &=& \underbrace{00...00}_{j_1}\ 1\ \underbrace{00...00}_{n-j_1-1} \\ \underline{0} &=& \underbrace{00...00}_{j_1}\ 0\ \underbrace{00...00}_{n-j_1-1} \\ S &=& \underbrace{00.1.00}_{j_1}\ 0\ \underbrace{00...00}_{n-j_1-1} \end{array} \quad \text{or} \quad \begin{array}{rcl} R &=& \underbrace{00...00}_{j_1}\ 1\ \underbrace{00...00}_{n-j_1-1} \\ \underline{0} &=& \underbrace{00...00}_{j_1}\ 0\ \underbrace{00...00}_{n-j_1-1} \\ S &=& \underbrace{00...00}_{j_1}\ 0\ \underbrace{00.1.00}_{n-j_1-1} \end{array}$$

There are n possible values of R, and for each, there are $n-1$ possible values of S. Hence, the sum of children for this case is

$$C_2 := n^2 - n \tag{6}$$

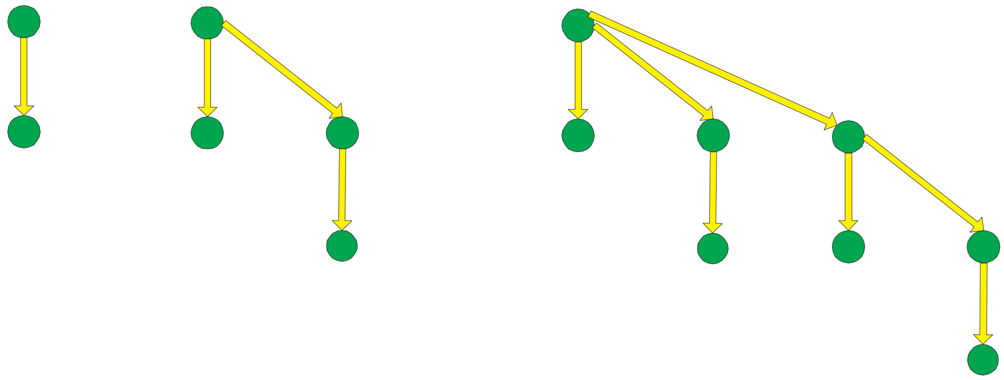


Figure 13: Binomial Trees.

Again, to support our claim that no other node has more children, one can verify that all nodes $\neq \underline{0}$ cannot have more children in this case.
Case 3: If $\underline{0}$ itself is the root, the sum of children is directly obtained as

$$C_3 := n \tag{7}$$

Taking the average over all three we can write down the solution for w_{max}.

$$\begin{aligned} w_{max} &= \frac{C_1 + C_2 + C_3}{N} && (8)\\ &= \frac{1}{N}\left[(2N - n - 2 - n^2) + (n^2 - n) + n\right] && (9)\\ &= 2 - \frac{n+2}{N} && (10) \end{aligned}$$

A.2 Number of Descendants

A.2.1 Total Average Number of Descendants v_{avg}

Consider the binomial trees for $n = 1, 2, 3$ shown in Figure 13, obtained when executing our embedding algorithm for a hypercube with n dimensions. Let us introduce $W^{(k)}$ to denote the sum of descendants of all nodes in a binomial tree of size k. For $k = 1$ we see by inspecting Figure 13 that the total sum of descendants is 3: the root node has two descendants and the child node has one descendant. Therefore,

$$W^{(1)} = 3 \tag{11}$$

Since a binomial tree of size k is obtained by taking two binomial trees with size $k-1$ and interconnecting them at the root (see Figure 13), we obtain $W^{(k)}$ from $W^{(k-1)}$ as follows:

$$W^{(k)} = 2W^{(k-1)} + 2^{k-1} \tag{12}$$

We can resolve the recursion in Eqns. (11) and (12) to

$$W^{(k)} = 2^k + k2^{k-1} \tag{13}$$

Since v_{avg} is simply $W^{(k)}/2^k$ and with $N = 2^n$ we obtain

$$\begin{aligned} v_{avg} = W^{(n)}/2^n &= \frac{1}{2^n}\left[2^n + n2^{n-1}\right] && (14) \\ &= \frac{n}{2} + 1 && (15) \end{aligned}$$

A.2.2 Maximum Average Number of Descendants v_{max}

We claim that the maximum average occurs for the node with label address 00..0, and call this node $\underline{0}$. (We do not provide a proof, however, it is not hard to see that in each of the cases that we will consider, no node can have more descendants than node $\underline{0}$.) Proceeding in a similar fashion as for the proof of maximum average number of children in Subsection A.1.2, we address the question of calculating, for any embedded tree with root R, the number of descendants of node 00...0, or $\underline{0}$. Again we consider three different scenarios:

Case 1: The label of R contains two or more 1's.

Case 2: The label of R contains a single 1.

Case 3: The label of R is all 0's, i.e., $\underline{0}$ is the root.

<u>**Case 1:**</u> The root has the following structure, with j_1 leading zeroes and j_2 trailing zeroes.

$$R = \underbrace{00...00}_{j_1}\ 1\ xx\ldots xx\ 1\ \underbrace{00...00}_{j_2}$$

Let S be a descendant of $\underline{0}$ in the embedded tree with R as root. Then S must have all 0's in the middle portion, and have 1's in only the leading and trailing portions as shown below (again, an 'x' indicates either a '1' or a '0').

$$\begin{aligned} R &= \underbrace{00...00}_{j_1}\ 1\ xx...xx\ 1\ \underbrace{00...00}_{j_2} \\ \underline{0} &= \underbrace{00...00}_{j_1}\ 0\ 00\ldots 00\ 0\ \underbrace{00...00}_{j_2} \\ S &= \underbrace{xx...xx}_{j_1}\ 0\ 00...00\ 0\ \underbrace{xx...xx}_{j_2} \end{aligned}$$

We see that for each value of j with $j = j_1 + j_2$, every node with $n - j$ 0's in the middle part is a descendant of $\underline{0}$. Thus, node $\underline{0}$ has 2^j descendants where $j = j_1 + j_2$. For each value of j, there are $(j+1)2^{n-j-2}$ different values of R. Thus, for each value of j, the total number of descendants that $\underline{0}$ has, is $(j+1)2^{n-2}$. Summing over all values of j, we have.

$$D_1 \quad := \quad \sum_{j=0}^{n-2} 2^j (j+1) 2^{n-j-2} \tag{16}$$

After some simplifications we obtain

$$D_1 \quad = \quad 2^{n-3} n(n-1) \tag{17}$$

<u>**Case 2:**</u> Now consider that root R has exactly one 1 in its label, with $j-1$ leading 0's.

$$R = \underbrace{00...00}_{j_1}\ 1\ \underbrace{00...00}_{n-j_1-1}$$

Here, the relationship between R, $\underline{0}$, and S is the following.

$$\begin{aligned} R &= \underbrace{00...00}_{j_1}\ 1\ \underbrace{00...00}_{n-j_1-1} \\ \underline{0} &= \underbrace{00...00}_{j_1}\ 0\ \underbrace{00...00}_{n-j_1-1} \\ S &= \underbrace{xx...xx}_{j_1}\ 0\ \underbrace{xx...xx}_{n-j_1-1} \end{aligned}$$

Note that there are n possible labels for R, each with a different position for the 1. For each of these, there are 2^{n-1} possible values of S. Hence, the total number of descendants in this case is

$$D_2 := n2^{n-1} \tag{18}$$

<u>**Case 3:**</u> Finally, we consider that $\underline{0}$ itself is the root. The number of descendants are all nodes in the embedded tree, that is

$$D_3 := 2^n \tag{19}$$

The average for $\underline{0}$ therefore is,

$$v_{max} \quad = \quad \frac{D_1 + D_2 + D_3}{N} \tag{20}$$

$$\quad = \quad \frac{2^{n-3}n(n-1) + n2^{n-1} + 2^n}{N} \tag{21}$$

With $N = 2^n$, we can simplify this expression to

$$v_{max} = \frac{n^2}{8} + \frac{3n}{8} + 1 \tag{22}$$

A.3 Path Length to the Root

By construction, the path length of a node in a tree that is embedded in the hypercube only depends on the Hamming distance between the root and the node in question. For any root node, the average path length is its average hamming distance to other nodes.

A.3.1 Total Average Path Length to the Root p_{avg}

We will show that for all trees it holds that $p_{avg} = v_{avg} - 1$.
By definition of p_{avg} we have

$$p_{avg} = \frac{1}{N^2} \sum_{x \in V} \sum_{r \in V} p_x(T_r) \tag{23}$$

Denote by $P_{x,r}$ the collection of nodes in a tree with root r that are on the path from x to r. With $P_{x,r}$ we can rewrite the previous equation as

$$\begin{aligned} p_{avg} &= \frac{1}{N^2} \left[\sum_{r \in V} \sum_{x \in V} \left(\left(\sum_{y \in P_{x,r}} 1 \right) - 1 \right) \right] & (24) \\ &= \frac{1}{N^2} \left[\sum_{r \in V} \sum_{x \in V} \left(\sum_{y \in P_{x,r}} 1 \right) \right] - 1 & (25) \end{aligned}$$

By exchanging the order of summation we can write

$$p_{avg} = \frac{1}{N^2} \left[\sum_{r \in V} \sum_{y \in V} \left(\sum_{x \,:\, y \in P_{x,r}} 1 \right) \right] - 1 \tag{26}$$

Noting that $\sum_{x \,:\, y \in P_{x,r}} 1 = v_y(T_r)$ is the number of descendants of node y in the tree with root r, we obtain

$$\begin{aligned} p_{avg} &= \frac{1}{N^2} \left[\sum_{r \in V} \sum_{y \in V} v_y(T_r) \right] - 1 & (27) \\ &= v_{ave} - 1 & (28) \end{aligned}$$

Inserting the result from Eqn. (15) yields the result

$$p_{avg} = \frac{n}{2} \tag{29}$$

A.3.2 Maximum Average Path Length to the Root p_{max}

In a complete hypercube, the average path length to the root is the same for all embedded trees. Hence, the average path length p_{avg} is the same as the maximum p_{max}.

$$p_{max} = p_{avg} = \frac{n}{2} \tag{30}$$

B The incomplete n-dimensional Hypercube ($N < 2^n$)

For brevity, we only derive a set of relatively loose bounds. Although these bounds are only crude approximations they exhibit the same growth behavior as the exact expressions for the full hypercube.

Let us use X_{avg} to denote either one of w_{avg}, v_{avg}, or p_{avg}, and X_{max} to denote either one of w_{max}, v_{max}, and p_{max}. Furthermore, let $X_{avg}^{(N)}$ and $X_{max}^{(N)}$ denote the respective values for a hypercube with N nodes. We will now derive bounds for $X_{avg}^{(N)}$ and $X_{max}^{(N)}$ for hypercubes with N nodes and $2^{n-1} < N < 2^n$.

Let us first consider $X_{max}^{(N)}$. Assuming that the maximum is attained for node s, that is, $X_s^{(N)} = \max_k X_k^{(N)}$, we have

$$X_{max}^{(N)} = \frac{1}{N} \sum_{r=1}^{N} X_s^{(N)}(T_r) \tag{31}$$

Clearly, we can increase the righthand side by adding terms.

$$\begin{aligned} X_{max}^{(N)} &< \frac{1}{N} \sum_{r=1}^{2^n} X_s^{(N)}(T_r) && (32) \\ &= \frac{1}{N} 2^n \left(\frac{1}{2^n} \sum_{r=1}^{2^n} X_s^{(N)}(T_r) \right) && (33) \\ &= \frac{2^n}{N} X_{max}^{(2^n)} && (34) \\ &< 2X_{max}^{(2^n)} && (35) \end{aligned}$$

In the same way, we can derive a lower bound and obtain

$$X_{max}^{(N)} > \frac{1}{2} X_{max}^{(2^{n-1})} \tag{36}$$

Now let us consider $X_{ave}^{(N)}$. With a similar derivation as above we can derive

$$\begin{aligned} X_{ave}^{(N)} &= \frac{1}{N^2} \sum_{k=1}^{N} \sum_{r=1}^{N} X_k^{(N)}(T_r) && (37) \\ &< \frac{1}{N^2} \sum_{k=1}^{2^n} \sum_{r=1}^{2^n} X_k^{(N)}(T_r) && (38) \\ &= \frac{4^n}{N^2} X_{ave}^{(2^n)} && (39) \\ &< 4X_{ave}^{(2^n)} && (40) \end{aligned}$$

The corresponding lower bound is

$$X_{ave}^{(N)} > \frac{1}{4} X_{ave}^{(2^{n-1})} \tag{41}$$

Focusing only on the upper bounds, we can use the results for complete hypercubes from Appendix A and obtain

$$w_{avg}^{(N)} < 4 - \frac{2}{N} \tag{42}$$

$$
\begin{aligned}
w_{max}^{(N)} &< 4 - \frac{\log_2 N + 3}{N} && (43)\\
v_{ave}^{(N)} &< 2\log_2 N + 6 && (44)\\
v_{max}^{(N)} &< \frac{1}{4}(\log_2 N)^2 + \frac{5}{4}\log_2 N + 3 && (45)\\
p_{avg}^{(N)} &< 2\log_2 N + 2 && (46)\\
p_{max}^{(N)} &< \log_2 N + 1 && (47)
\end{aligned}
$$

Some of the above bounds can be improved as follows:

$$
\begin{aligned}
w_{ave}^{(N)} &= 1 - \frac{1}{N} && && (48)\\
p_{avg}^{(N)} &\leq p_{max}^{(N)} &&< \log_2 N + 1 && (49)\\
v_{ave}^{(N)} &= p_{ave}^{(N)} + 1 &&\leq \log_2 N + 1 && (50)
\end{aligned}
$$

Eqn. (48) follows from the exact expression for w_{avg} from Subsection A.1.1. The tighter bound in Eqn. (49) is obvious as the average is never larger than the maximum. Eqn. (50) exploits that $v_{ave} = p_{ave} + 1$ as shown in Subsection A.3.1.

C The Shared K-ary Tree

C.1 Number Of Children

C.1.1 Total Average Number of Children w_{avg}

In Subsection A.1.1 we showed that the following holds for all trees:

$$w_{avg} = 1 - \frac{1}{N} \tag{51}$$

C.1.2 Maximum Average Number of Children w_{max}

The maximum $w_{max} = \max_k \{w_k\}$ is attained for a node in the original tree with maximum degree. (The degree of a node is the number edges connected to the node.) In a K-ary tree, the maximal degree of any node is $K+1$. If the tree is re-hung, a node with maximal degree has $K+1$ children if it is the root of the re-hung tree, and K children otherwise. Therefore,

$$w_{max} = K + \frac{1}{N} \tag{52}$$

C.2 Number of Descendants

Before calculating the values v_{max} and v_{avg}, we define E_l, the number of nodes in a subtree of a node which is at level l in the original tree. Since, by assumption, the node has K descendants at level $l+1$, K^2 descendants at level $l+2$, etc., a simple geometric progression yields

$$E_l = 1 + K + K^2 + ... + K^{d-l} = \frac{K^{d-l+1} - 1}{K - 1} \tag{53}$$

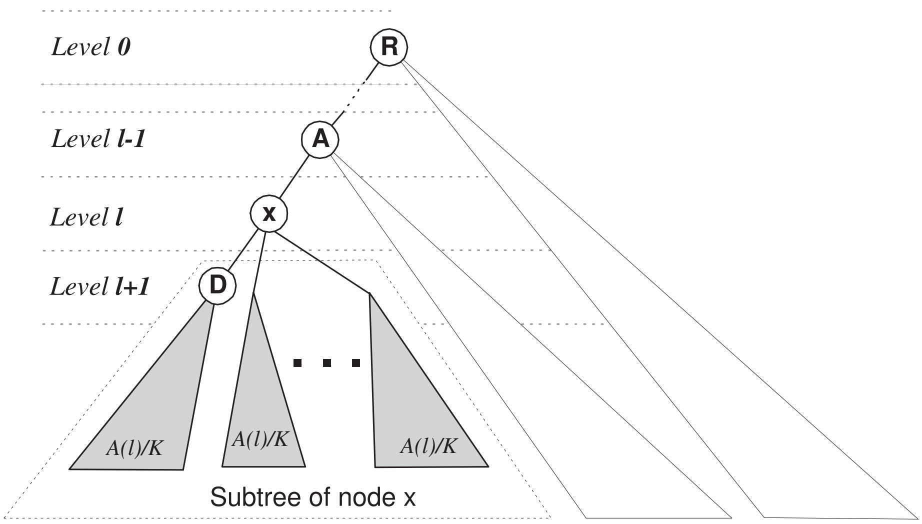


Figure 14: Structuring of Subtrees for Calculation of v_l.

C.2.1 Total Average Number of Descendants v_{avg}

We now use E_l from Eqn. (53) to calculate the average number of descendants of a node averaged over all re-hung trees. With a light abuse of notation, we denote by v_l the average number of descendants for a node which is at level l in the original tree.

Let us consider a K-ary tree ('original tree') and tag a node at level l. Such a tree is shown in Figure 14. Here, the tagged node is labeled as *node x*, its ancestor node is *node A*, one of its children is labeled as *node D*, and the root node is labeled *node R*. When such a tree is re-hung, three types of trees may result:

1. *The tagged* node x *is the root in the re-hung tree (see Figure 15(a)).*
   Here, the tagged node has N descendants. Among all N possible re-hung trees, there is only one tree which has the tagged node as root.

2. *The root node of the re-hung tree was a descendant of the tagged node in the original tree. (For example, the root is* node D *or a descendant of* node D*).*
   There is one re-hung tree for each of the $E_l - 1$ descendants of *node x*. One such tree is depicted in Figure 15(b). In each tree, the number of descendants of node *node x* consists of all nodes in the tree except the nodes in the subtree of *node x* (in the original tree) which contains the root in the re-hung tree. The number of descendants is $N-(E_l-1)/K$ nodes.

3. *The root of the re-hung tree is neither the tagged node nor a descendant of the tagged node in the original tree.*

There are a total of $N - E_l$ such re-hung trees. Here, the descendants of *node* x in the re-hung tree are the descendants of *node* x in the original tree.

With these considerations, we obtain v_l by averaging over all N trees

$$\begin{aligned} v_l &= \frac{1}{N}\left[N + (E_l - 1)(N - \frac{E_l - 1}{K}) + E_l\,(N - E_l)\right] && (54) \\ &= \frac{1}{N}\left[2E_l\left(N + \frac{1}{K}\right) - (E_l)^2\left(1 + \frac{1}{K}\right) - \frac{1}{K}\right] && (55) \end{aligned}$$

Now we obtain v_{avg} by averaging v_l over all levels, that is,

$$v_{avg} = \frac{1}{N}\sum_{l=0}^{d} K^l v_l \qquad (56)$$

With Eqn. (55) and some algebraic manipulations we get

$$v_{avg} = 2d + \frac{K-5}{K-1} + \frac{6d}{N(K-1)} + \frac{4(K-2)}{N(K-1)^2} + \frac{4(d+1)}{N^2(K-1)^2} \qquad (57)$$

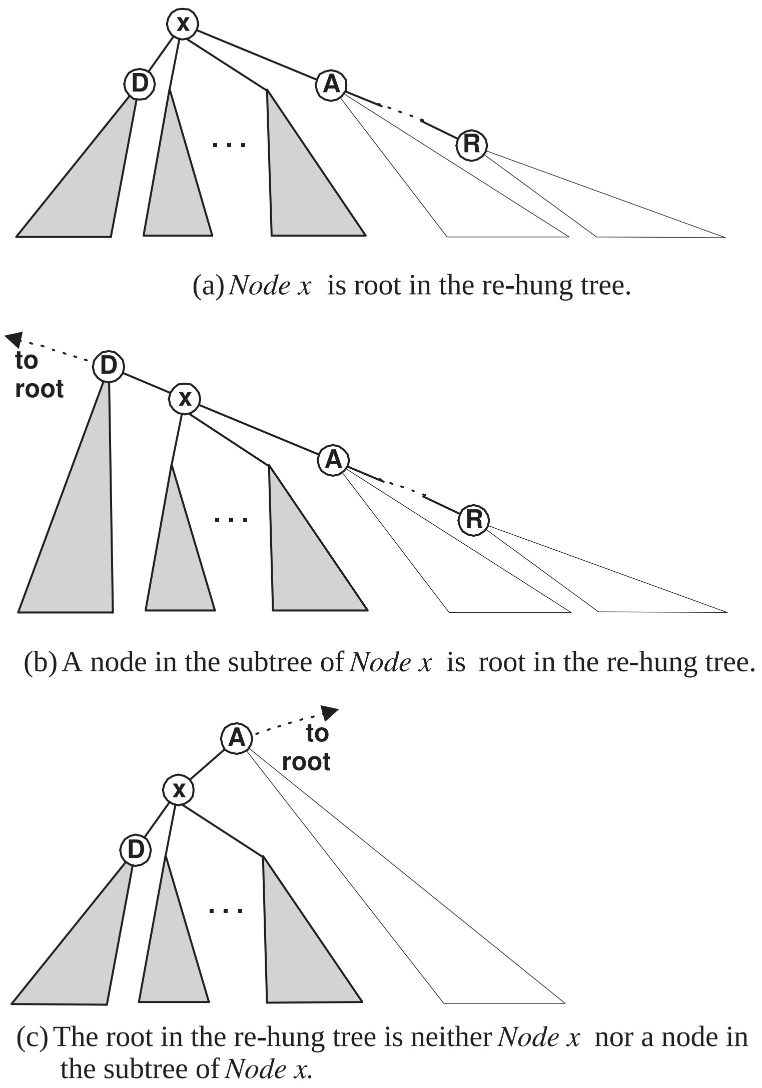
C.2.2 Maximum Average Number of Descendants v_{max}

Not completely intuitively, the maximum average number of descendants is not always attained for the root node of the original tree. Taking the maximum in Eqn. (55) we calculate for sufficiently large N the level l_{max} with $v_{l_{max}} = \max_l v_l$ as

$$l_{max} = \begin{cases} 1 & \text{if } K = 2 \\ 0 & \text{otherwise} \end{cases} \qquad (58)$$

Inserting l_{max} into Eqn. (55) we obtain

$$v_{max} = v_{l_{max}} = \begin{cases} \dfrac{5N}{8} + \dfrac{1}{4} - \dfrac{11}{8N} & \text{if } K = 2 \\[2ex] \dfrac{(K-1)N}{K} + \dfrac{1}{KN} & \text{otherwise} \end{cases} \qquad (59)$$



(a) *Node x* is root in the re-hung tree.

(b) A node in the subtree of *Node x* is root in the re-hung tree.

(c) The root in the re-hung tree is neither *Node x* nor a node in the subtree of *Node x*.

Figure 15: Re-hung Trees as Used for the Derivation of v_l.

C.3 Path Length to the Root

C.3.1 Total Average Path Length to the Root p_{ave}

Since we derived in Subsection A.3.1 that $p_{avg} = v_{avg} - 1$ holds for all trees, inserting Eqn. (57) yields the result

$$p_{avg} = 2d - \frac{4}{K-1} + \frac{6d}{N(K-1)} + \frac{4(K-2)}{N(K-1)^2} + \frac{4(d+1)}{N^2(K-1)^2} \tag{60}$$

C.3.2 Maximum Average Path Length to the Root p_{max}

Let p_l denote the average path length to the root of a tagged noded at level l in the original tree, averaged over all positions of the root. (Again, we abuse notation and use the index to indicate the level of a node.) For $l = 0$ we have

$$p_0 = \frac{1}{N}\sum_{l=0}^{d} lK^l \tag{61}$$

$$= d - \frac{1}{K-1} + \frac{d+1}{N(K-1)} \tag{62}$$

For $l > 0$, we can derive a relationship between p_l and p_{l-1}. Consider the tree shown in Figure 14. Let us tag *node* x which is at level l. To obtain p_l we calculate the average path length from *node* x to the roots in the re-hung trees. Let us assume that we know p_{l-1}, i.e., the average path length of a parent of *node* x (which is *node* A in Figure 14). We can use p_{l-1} to obtain the average path length if *node* x is the tagged node. If the root is in the subtree of *node* x, including *node* x itself, then the path from *node* x to the root is one edge shorter, than the path from *node* A to the root. Otherwise, the path is one edge longer. Therefore, we can write

$$p_l = \frac{1}{N}\left[Np_{l-1} - E_l + (N - E_l)\right] \tag{63}$$

$$= p_{l-1} - \frac{2}{N}\frac{K^{d-l+1}-1}{K-1} + 1 \tag{64}$$

Resolving the recursive expression yields

$$p_l = p_0 - \frac{2}{N(K-1)}[N - E_l - l] + l \tag{65}$$

Since the average path length monotonically increases in l, the maximum occurs at $l = d$, i.e.,

$$p_{max} = p_d = p_0 - \frac{2}{N(K-1)}[N - E_d - d] + d \tag{66}$$

With Eqn. (62) and $E_d = 1$, we obtain

$$p_{max} = 2d - \frac{3}{K-1} + \frac{3(d+1)}{N(K-1)} \tag{67}$$